Symfony 5 : API REST

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en Programmation par contrainte (IA)
Ingénieur en Génie logiciel

elmouelhi.achref@gmail.com

Plan

1. Introduction

2. Contrôleur avec réponse JSON
   - `NormalizerInterface`
   - `@Groups`
   - `SerializerInterface`
   - `JsonResponse`
   - La méthode `json`

Plan

3 API Platform
- `@ApiResource`
- Collection et item operations
- `routePrefix`
- Pagination
- `normalizationContext` et `denormalizationContext`
- `DataPersisterInterface`
- `@ApiSubresource`
- `@ApiFilter`

4 JWT et API Platform

Symfony

Service web (WS pour Web Service) ?

- Un programme (ensemble de fonctionnalités exposées en temps réel et sans intervention humaine)
- Accessible via internet, ou intranet
- Indépendant de tout système d'exploitation
- Indépendant de tout langage de programmation
- Utilisant un système standard d'échange (**XML** ou **JSON**), ces messages sont généralement transportés par des protocoles internet connus **HTTP** (ou autres comme **FTP**, **SMTP**...)
- Pouvant communiquer avec d'autres WS

Symfony

Les WS peuvent utiliser les technologies web suivantes :

- **HTTP** (Hypertext Transfer Protocol) : le protocole, connu, utilisé par le World Wide Web et inventé par Roy Fiedling.
- **REST** (Representational State Transfer) : une architecture de services Web, créée aussi par Roy Fielding en 2000 dans sa thèse de doctorat.
- **SOAP** (Simple object Access Protocol) : un protocole, défini par Microsoft et IBM ensuite standarisé par **W3C**, permettant la transmission de messages entre objets distants (physiquement distribués).
- **WSDL** (Web Services Description Language) : est un langage de description de service web utilisant le format **XML** (standardisé par le **W3C** depuis 2007).
- **UDDI** (Universal Description, Discovery and Integration) : un annuaire de WS.

HTTP, **REST** (Representational State Transfer) et **RESTful** ?

- Les API REST sont basées sur le protocole **HTTP** (architecture client/serveur) et utilisent le concept de ressource.
- Une ressource est identifiée par une URI unique.
- L'API REST utilise donc des méthodes suivantes pour l'échange de données entre client et serveur
  - `GET` pour la récupération,
  - `POST` pour l'ajout,
  - `DELETE` pour la suppression,
  - `PUT` pour la modification,
  - ...
- Plusieurs formats possibles pour les données échangées : texte, **XML**, **JSON**...
- **RESTful** est l'adjectif qui désigne une API REST.

Symfony

Rôle du contrôleur dans une application **MVC**

- Le contrôleur reçoit une requête **HTTP** et communique avec modèle, service, constructeur de formulaires pour retourner une réponse **HTTP** contenant une page **HTML**.
- Le contrôleur peut aussi retourner une réponse **HTTP** ne contenant pas de vue.
- Il retourne des données sous format **JSON**, **XML**...

Symfony

Rôle du contrôleur dans une application **MVC**

- Le contrôleur reçoit une requête **HTTP** et communique avec modèle, service, constructeur de formulaires pour retourner une réponse **HTTP** contenant une page **HTML**.
- Le contrôleur peut aussi retourner une réponse **HTTP** ne contenant pas de vue.
- Il retourne des données sous format **JSON**, **XML**...

Ceci est l'objet de ce chapitre.

Symfony

Étapes à suivre avant de commencer le cours

1. Allez à `https://github.com/Achreftun/api-rest-symfony`
2. Clonez le dépôt dans votre espace de travail
3. Installez les dépendances : `composer install` ou `composer update`
4. Créez la base de données : `php bin/console d:d:c`
5. Appliquez les migrations : `php bin/console d:m:m`
6. Envoyez les fixtures (les tuples) à la base de données : `php bin/console d:f:l`
7. Lancez le server : `symfony server:start`

Symfony

Utilisation de **Postman** : simulateur d'un client REST

- Aller sur internet et chercher **Postman**
- Télécharger puis installer l'application **Postman**
- Démarrer l'application

Il existe plusieurs autres tel que ARC (pour Advanced REST Client)...

Symfony

Créons un contrôleur `ApiPersonneController`

- Exécutez `php bin/console make:controller`
- Répondez à `Choose a name for your controller class` par `ApiPersonne`

Symfony

Contenu généré de `ApiPersonneController`

```
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;

class ApiPersonneController extends AbstractController
{
    /**
     * @Route("/api/personne", name="api_personne")
     */
    public function index()
    {
        return $this->json([
            'message' => 'Welcome to your new controller!',
            'path' => 'src/Controller/ApiPersonneController.php',
        ]);
    }
}
```

Pour retourner les tuples de la table `Personne` sous format `JSON`

```
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use App\Repository\PersonneRepository;

class ApiPersonneController extends AbstractController
{
    /**
     * @Route("/api/personne", name="api_personne")
     */
    public function index(PersonneRepository $personneRepository)
    {
        $personnes = $personneRepository->findAll();
        $json = json_encode($personnes);
        return $json;
    }
}
```

Pour retourner les tuples de la table `Personne` sous format `JSON`

```php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use App\Repository\PersonneRepository;

class ApiPersonneController extends AbstractController
{
    /**
     * @Route("/api/personne", name="api_personne")
     */
    public function index(PersonneRepository $personneRepository)
    {
        $personnes = $personneRepository->findAll();
        $json = json_encode($personnes);
        return $json;
    }
}
```

Erreur

```
The controller must return a Symfony\Component\HttpFoundation\Response
object but it returned a string
```

Pour retourner une réponse HTTP contenant les tuples sous format JSON

```
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Response;
use App\Repository\PersonneRepository;

class ApiPersonneController extends AbstractController
{
    /**
     * @Route("/api/personne", name="api_personne")
     */
    public function index()
    {
        $personnes = $personneRepository->findAll();
        $json = json_encode($personnes);
        $reponse = new Response($json, 200, [
            'content-type' => 'application/json'
        ]);
        return $reponse;
    }
}
```

Pour retourner une réponse HTTP contenant les tuples sous format `JSON`

```
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Response;
use App\Repository\PersonneRepository;

class ApiPersonneController extends AbstractController
{
    /**
     * @Route("/api/personne", name="api_personne")
     */
    public function index()
    {
        $personnes = $personneRepository->findAll();
        $json = json_encode($personnes);
        $reponse = new Response($json, 200, [
            'content-type' => 'application/json'
        ]);
        return $reponse;
    }
}
```

Objets JSON vides car les attributs sont privés.

Symfony

Solution ⇒ normaliser ou serializer les objets. Pour le faire

- Installez le bundle de serialisation
- Configurez `config/services.yaml`

Symfony

Solution ⇒ normaliser ou serializer les objets. Pour le faire

- Installez le bundle de serialisation
- Configurez `config/services.yaml`

Pour installer le bundle de serialisation

```
composer require symfony/serializer
```

Symfony

Solution ⇒ normaliser ou serializer les objets. Pour le faire

- Installez le bundle de serialisation
- Configurez `config/services.yaml`

Pour installer le bundle de serialisation

```
composer require symfony/serializer
```

Dans `services.yaml`, ajoutez le code suivant pour qu'on puisse utiliser les getters/setters pour accéder aux attributs privés

```
services:
    get_set_method_normalizer:
        class: Symfony\Component\Serializer\Normalizer\
          GetSetMethodNormalizer
        tags: [serializer.normalizer]
```

Symfony

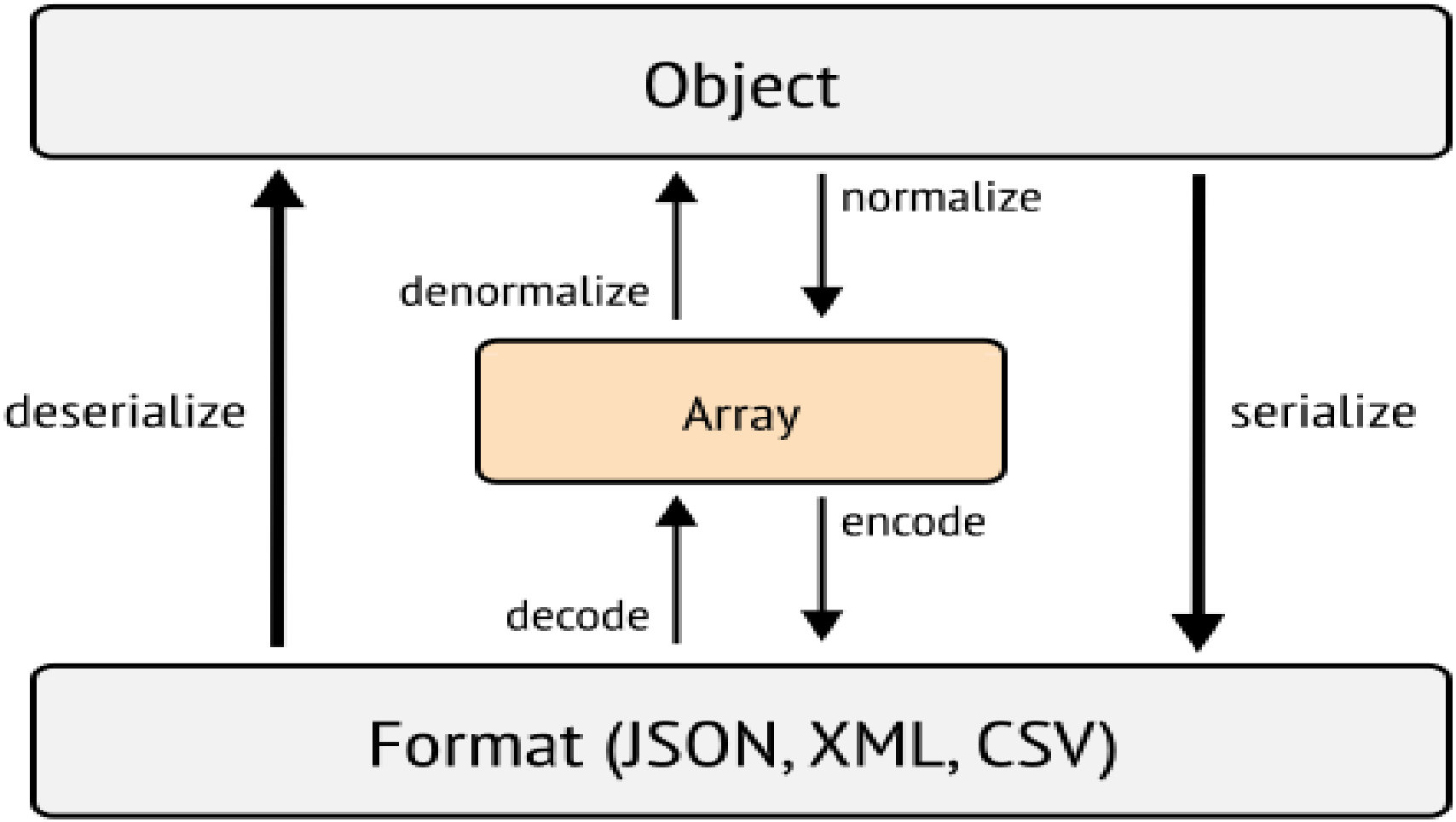


Image de la documentation officielle de Symfony

Pour retourner une réponse HTTP contenant les tuples sous format JSON

```
namespace App\Controller;

use App\Repository\PersonneRepository;
use Symfony\Component\Serializer\Normalizer\NormalizerInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class ApiPersonneController extends AbstractController
{
    /**
     * @Route("/api/personne", name="api_personne", methods="GET")
     */
    public function index(PersonneRepository $personneRepository,
      NormalizerInterface $normalizer)
    {
        $personnes = $personneRepository->findAll();
        $normalized = $normalizer->normalize($personnes);
        $json = json_encode($normalized);
        return new Response($json, 200, [
            'content-type' => 'application/json'
        ]);
    }
}
```

Symfony

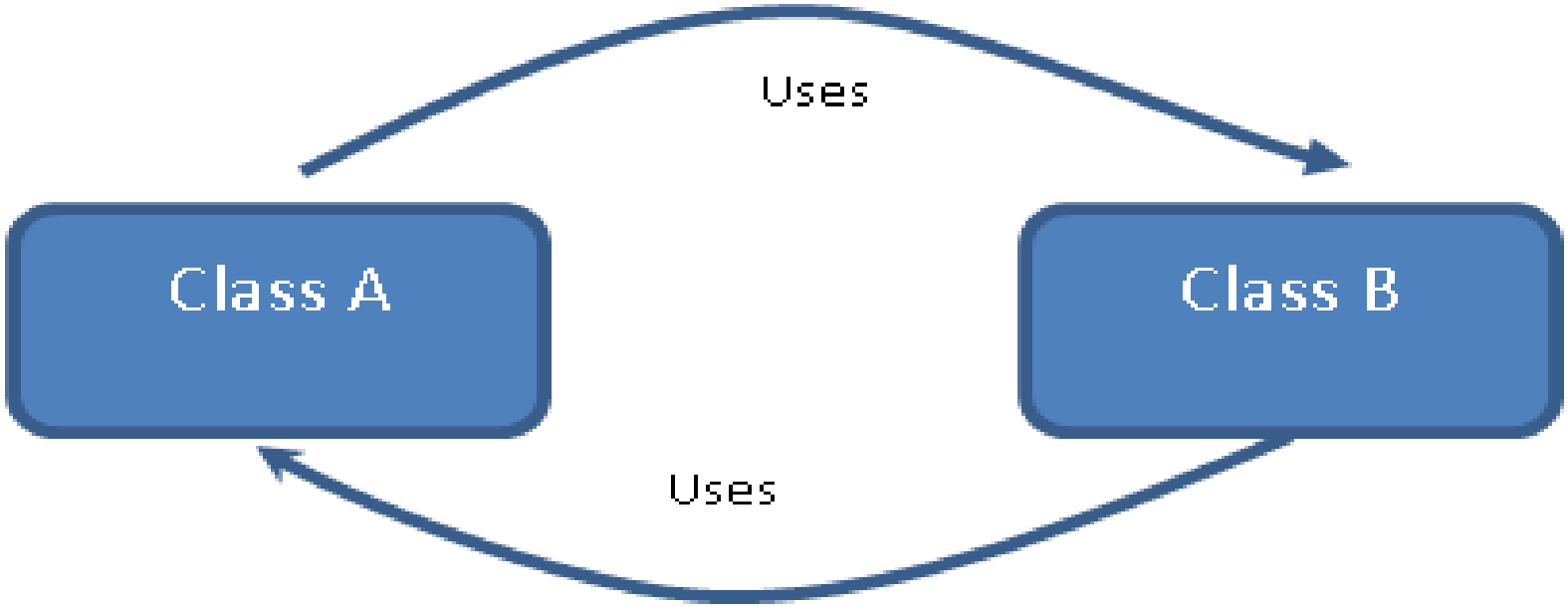
En renvoyant la requête HTTP depuis **POSTMAN**

```
A circular reference has been detected when
serializing the object of class
\"App\\Entity\\Personne\"
```

Symfony

En renvoyant la requête HTTP depuis **POSTMAN**

```
A circular reference has been detected when
serializing the object of class
\"App\\Entity\\Personne\"
```



Pour résoudre ce problème, on doit créer des groupes dans l'entité `Personne`

```
class Personne
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     * @Groups("personne:read")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups("personne:read")
     */
    private $nom;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups("personne:read")
     */
    private $prenom;

    /**
     * @ORM\ManyToMany(targetEntity=Adresse::class, inversedBy="personnes", cascade={"remove",
        "persist"})
     */
    private $adresses;
```

Pour résoudre ce problème, on doit créer des groupes dans l'entité `Personne`

```
class Personne
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     * @Groups("personne:read")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups("personne:read")
     */
    private $nom;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups("personne:read")
     */
    private $prenom;

    /**
     * @ORM\ManyToMany(targetEntity=Adresse::class, inversedBy="personnes", cascade={"remove",
        "persist"})
     */
    private $adresses;
```

Le `use` nécessaire

```
use Symfony\Component\Serializer\Annotation\Groups;
```

Symfony

En utilisant la méthode `normalize`, on indique le·s groupe·s à inclure dans la réponse

```
class ApiPersonneController extends AbstractController
{
    /**
     * @Route("/api/personne", name="api_personne", methods="GET")
     */
    public function index(PersonneRepository $personneRepository,
      NormalizerInterface $normalizer)
    {
        $personnes = $personneRepository->findAll();
        $normalized = $normalizer->normalize($personnes, null, [
            'groups' => 'personne:read'
        ]);

        $json = json_encode($normalized);

        return new Response($json, 200, [
            'content-type' => 'application/json'
        ]);
    }
}
```

Et si on voulait retourner pour chaque objet `Personne` sa liste `adresses`, on ajoute le groupe `personne:read` à `adresses` dans `Personne`

```
class Personne
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     * @Groups("personne:read")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups("personne:read")
     */
    private $nom;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups("personne:read")
     */
    private $prenom;

    /**
     * @ORM\ManyToMany(targetEntity=Adresse::class, inversedBy="personnes", cascade={"remove",
        "persist"})
     * @Groups("personne:read")
     */
    private $adresses;
```

Et dans `Adresse`, on ajoute `personne:read` pour tous les attributs sauf `personnes`

```php
class Adresse
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     * @Groups("personne:read")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups("personne:read")
     */
    private $rue;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups("personne:read")
     */
    private $codePostal;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups("personne:read")
     */
    private $ville;

    /**
     * @ORM\ManyToMany(targetEntity=Personne::class, mappedBy="adresses")
     */
    private $personnes;
```

On peut aussi utiliser l'interface `SerializerInterface` qui normalise et encode en JSON

```
namespace App\Controller;

use App\Repository\PersonneRepository;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Serializer\SerializerInterface;

class ApiPersonneController extends AbstractController
{
    /**
     * @Route("/api/personne", name="api_personne", methods="GET")
     */
    public function index(PersonneRepository $personneRepository,
      SerializerInterface $serializer)
    {
        $personnes = $personneRepository->findAll();
        $json = $serializer->serialize($personnes, 'json', ['groups' =>
            'personne:read']);
        return new Response($json, 200, [
            'content-type' => 'application/json'
        ]);
    }
}
```

Pour la réponse, on peut simplifier le code en utilisant un objet `JsonResponse`

```php
namespace App\Controller;

use App\Repository\PersonneRepository;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Serializer\SerializerInterface;

class ApiPersonneController extends AbstractController
{
    /**
     * @Route("/api/personne", name="api_personne", methods="GET")
     */
    public function index(PersonneRepository $personneRepository,
      SerializerInterface $serializer)
    {
        $personnes = $personneRepository->findAll();
        $json = $serializer->serialize($personnes, 'json', [
            'groups' => 'personne:read'
        ]);
        return new JsonResponse($json, 200,  [], true);
    }
}
```

Symfony

On peut aussi simplifier l'écriture en utilisant la méthode `json` de `AbstractController` qui retourne un objet `JsonResponse`

```php
namespace App\Controller;

use App\Repository\PersonneRepository;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;

class ApiPersonneController extends AbstractController
{
    /**
     * @Route("/api/personne", name="api_personne", methods="GET")
     */
    public function index(PersonneRepository $personneRepository)
    {
        $personnes = $personneRepository->findAll();
        return $this->json($personnes, 200,  [], [
            'groups' => 'personne:read'
        ]);
    }
}
```

Symfony

Pour ajouter une nouvelle personne

```
/**
 * @Route("/api/personne", name="api_personne_add", methods="POST")
 */
public function add(EntityManagerInterface $entityManager, Request $request,
   SerializerInterface $serializer, ValidatorInterface $validator) {

    $contenu = $request->getContent();
    try {
        $personne = $serializer->deserialize($contenu, Personne::class, 'json');

        $errors = $validator->validate($personne);
        if (count($errors) > 0) {
            return $this->json($errors, 400);
        }

        $entityManager->persist($personne);
        $entityManager->flush();
        return $this->json($personne, 201, [], [
            'groups' => 'personne:read'
        ]);

    } catch (NotEncodableValueException $e) {
        return $this->json([
            'status' => 400,
            'message' => $e->getMessage()
        ]);
    }
}
```

Symfony

Les `use` nécessaires

```
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Serializer\SerializerInterface;
use Symfony\Component\Serializer\Exception\
   NotEncodableValueException;
use Symfony\Component\Validator\Validator\
   ValidatorInterface;
```

Symfony

Les `use` nécessaires

```
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Serializer\SerializerInterface;
use Symfony\Component\Serializer\Exception\
   NotEncodableValueException;
use Symfony\Component\Validator\Validator\
   ValidatorInterface;
```


Pensez à installer le `Validator`

```
composer require symfony/validator
```

Symfony

Pour tester avec **POSTMAN**

- Dans la liste déroulante, choisir `POST` puis saisir l'url vers notre web service `http://localhost:8000/api/personne`
- Dans le `Headers`, saisir `Content-Type` comme `Key` et `application/json` comme `Value`
- Ensuite cliquer sur `Body`, cocher `raw`, choisir `JSON (application/json)` et saisir des données sous format `json` correspondant à l'objet personne à ajouter

```
{
    "nom": "Morena",
    "prenom": "Andreas"
}
```

- Cliquer sur `Send`

Symfony

Exercice 1

Créez puis testez, avec **POSTMAN**, les deux méthodes **HTTP** `put` et `delete` qui permettront de modifier ou supprimer une personne.

Symfony

API Platform

- Framework **PHP** open-source
- Créé par Kévin Dunglas et distribué par **Symfony**
- Permettant de générer des services **REST** pour des entités **Doctrine**
- Utilisant **Swagger UI** (`https://swagger.io/tools/swagger-ui/`) pour visualiser et interagir avec les **API REST**

Symfony

API Platform

- Framework **PHP** open-source
- Créé par Kévin Dunglas et distribué par **Symfony**
- Permettant de générer des services **REST** pour des entités **Doctrine**
- Utilisant **Swagger UI** (`https://swagger.io/tools/swagger-ui/`) pour visualiser et interagir avec les **API REST**

Installation avec **Flex**

composer require api

Symfony

Contenu de `config/routes/api_platform.yaml`

```
api_platform:
    resource: .
    type: api_platform
    prefix: /api
```

Symfony

Contenu de `config/routes/api_platform.yaml`

```
api_platform:
    resource: .
    type: api_platform
    prefix: /api
```


Pour éviter le conflit avec nos travaux précédents, remplaçons `prefix: /api` par `prefix: /ws`

Symfony

Contenu de `config/routes/api_platform.yaml`

```
api_platform:
    resource: .
    type: api_platform
    prefix: /api
```

Pour éviter le conflit avec nos travaux précédents, remplaçons `prefix: /api` par `prefix: /ws`

Commentez ou supprimez les lignes ajoutées précédemment dans `config/serices.yaml`

```
    # get_set_method_normalizer:
    #     class: Symfony\Component\Serializer\Normalizer\
      GetSetMethodNormalizer
    #     tags: [serializer.normalizer]
```

Symfony

Ajoutons l'annotation `@ApiResource` à notre entité `Personne`

```
namespace App\Entity;

use App\Repository\PersonneRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Annotation\ApiResource;

/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResource
 */
class Personne
{
```

Symfony

Ajoutons l'annotation `@ApiResource` à notre entité `Personne`

```
namespace App\Entity;

use App\Repository\PersonneRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Annotation\ApiResource;

/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResource
 */
class Personne
{
```

La même chose pour l'entité `Adresse`.

Symfony

Pour tester

Allez sur `localhost:8000/ws`

Symfony

Deux types d'opérations

- Collection operations :
  - GET : `/ws/personnes`
  - POST : `/ws/personnes`
- Item operations :
  - GET : `/ws/personnes/{id}`
  - DELETE : `/ws/personnes/{id}`
  - PUT : `/ws/personnes/{id}`
  - PATCH : `/ws/personnes/{id}`

Symfony

Et si nous voulions exposer seulement quelques méthodes

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResource(
 *      collectionOperations={"get"},
 *      itemOperations={"get", "put", "delete"}
 * )
 */
class Personne
```

Symfony

Et si nous voulions exposer seulement quelques méthodes

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResource(
 *      collectionOperations={"get"},
 *      itemOperations={"get", "put", "delete"}
 * )
 */
class Personne
```

Les méthodes disponibles

- GET : `/ws/personnes`
- GET : `/ws/personnes/{id}`
- DELETE : `/ws/personnes/{id}`
- PUT : `/ws/personnes/{id}`

Symfony

Il est possible de modifier le `path` d'une méthode et le code de la réponse HTTP

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::
    class)
 * @ApiResource(
 *      collectionOperations={
 *           "get"={"path"="/person", "status"=301}
 *       },
 *      itemOperations={"get", "put", "delete"}
 * )
 */
class Personne
```

Symfony

Il est possible de modifier le `path` d'une méthode et le code de la réponse HTTP

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::
    class)
 * @ApiResource(
 *      collectionOperations={
 *           "get"={"path"="/person", "status"=301}
 *       },
 *      itemOperations={"get", "put", "delete"}
 * )
 */
class Personne
```

Pour tester `localhost:8000/ws/person`

Symfony

Pour modifier de préfixer le chemin de toutes les routes d'une ressource

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::
    class)
 * @ApiResource(
 *      routePrefix="/library"
 *      collectionOperations={
 *          "get"={"path"="/person", "status"=301}
 *      },
 *      itemOperations={"get", "put", "delete"}
 * )
 */
class Personne
```

Symfony

Pour modifier de préfixer le chemin de toutes les routes d'une ressource

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::
    class)
 * @ApiResource(
 *      routePrefix="/library"
 *      collectionOperations={
 *          "get"={"path"="/person", "status"=301}
 *      },
 *      itemOperations={"get", "put", "delete"}
 * )
 */
class Personne
```

Pour tester `localhost:8000/ws/library/person`

Symfony

Pour définir le nombre d'objets à retourner par requête (page)

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::
    class)
 * @ApiResource(
 *      attributes={"pagination_items_per_page"=3}
 *      collectionOperations={
 *           "get"={"path"="/person", "status"=301}
 *      },
 *      itemOperations={"get", "put", "delete"}
 * )
 */
class Personne
```

Symfony

Pour définir le nombre d'objets à retourner par requête (page)

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::
    class)
 * @ApiResource(
 *      attributes={"pagination_items_per_page"=3}
 *      collectionOperations={
 *           "get"={"path"="/person", "status"=301}
 *      },
 *      itemOperations={"get", "put", "delete"}
 * )
 */
class Personne
```

Pour tester `localhost:8000/ws/person?page=1`

Symfony

Objectif

- Ajouter un attribut `dateEnregistrement` qui correspond à la date d'insertion de la personne dans la base de données
- La valeur de ce champs ne doit pas être renseigné par l'utilisateur
- On utilise donc les `Groups` et les deux attributs `normalizationContext` et `denormalizationContext`

Nouveau contenu après avoir ajouté l'attribut `dateEnregistrement` et le groupe `personne:write`

```
class Personne
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     * @Groups({"personne:read", "personne:write"})
     */
    private $id;
    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups({"personne:read", "personne:write"})
     */
    private $nom;
    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups({"personne:read", "personne:write"})
     */
    private $prenom;
    /**
     * @ORM\ManyToMany(targetEntity=Adresse::class, inversedBy="personnes", cascade={"remove",
        "persist"})
     * @Groups({"personne:read", "personne:write"})
     */
    private $adresses;
    /**
     * @ORM\Column(type="datetime")
     * @Groups("personne:read")
     */
    private $dateEnregistrement;
```

Ajoutons les attributs suivants à l'annotation `@ApiResource`

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResource(
 *      itemOperations={"get", "put", "delete"},
 *      normalizationContext={"groups"={"personne:read"}},
 *      denormalizationContext={"groups"={"personne:write"}}
 * )
 */
class Personne
```

Ajoutons les attributs suivants à l'annotation `@ApiResource`

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResource(
 *      itemOperations={"get", "put", "delete"},
 *      normalizationContext={"groups"={"personne:read"}},
 *      denormalizationContext={"groups"={"personne:write"}}
 * )
 */
class Personne
```

Explication

- Les attributs ayant le groupe `"personne:read"` seront accessibles en lecture (`GET`)
- Les attributs ayant le groupe `"personne:write"` seront accessibles en mode écriture (`POST`, `PUT` et `DELETE`)
- Les attributs ayant les deux groupes seront accessibles en lecture et écriture
- Les attributs n'ayant aucun groupe ne seront pas pris en compte

Nouveau contenu de l'entité `Adresse`

```php
/**
 * @ORM\Entity(repositoryClass=AdresseRepository::class)
 * @ApiResource()
 */
class Adresse
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     * @Groups({"personne:read", "personne:write"})
     */
    private $id;
    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups({"personne:read", "personne:write"})
     */
    private $rue;
    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups({"personne:read", "personne:write"})
     */
    private $codePostal;
    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     * @Groups({"personne:read", "personne:write"})
     */
    private $ville;
    /**
     * @ORM\ManyToMany(targetEntity=Personne::class, mappedBy="adresses")
     */
    private $personnes;
```

Symfony

Remarques

- En essayant d'ajouter une nouvelle `Personne`, un message d'erreur s'affiche.
- En effet, la valeur de `dateEnregistrement` n'a pas été renseignée et elle n'est pas `nullable` pour **MySQL**.

Symfony

Remarques

- En essayant d'ajouter une nouvelle `Personne`, un message d'erreur s'affiche.
- En effet, la valeur de `dateEnregistrement` n'a pas été renseignée et elle n'est pas `nullable` pour **MySQL**.

Solution

Utiliser `DataPersister`

Symfony

Explication

- On crée une classe `PersonneDataPersister` qui implémente l'interface `DataPersisterInterface`
- `DataPersisterInterface` a trois méthodes :
  - `supports` : vérifie si le persister supporte l'entité
  - `persist` : s'exécute à chaque opération de type `POST` ou `PUT`
  - `remove` : s'exécute seulement pour l'opération `DELETE`

Contenu de la classe `PersonneDataPersister` **définie dans** `App/DataPersister`

```
namespace App\DataPersister;

use App\Entity\Personne;
use Doctrine\ORM\EntityManagerInterface;
use ApiPlatform\Core\DataPersister\DataPersisterInterface;

class PersonneDataPersister implements DataPersisterInterface
{
    private $entityManager;

    public function __construct(EntityManagerInterface $entityManager)
    {
        $this->entityManager = $entityManager;
    }
    public function supports($data, array $context = []): bool
    {
        return $data instanceof Personne;
    }
    public function persist($data, array $context = [])
    {
        $dateTimeZone = new \DateTimeZone('Europe/Paris');
        $data->setDateEnregistrement(new \DateTime('now', $dateTimeZone));
        $this->entityManager->persist($data);
        $this->entityManager->flush();
    }

    public function remove($data, array $context = [])
    {
        $this->entityManager->remove($data);
        $this->entityManager->flush();
    }
}
```

Symfony

Remarque

En faisant la modification d'une personne existante, `dateEnregistrement` sera aussi mise à jour

Symfony

Pour affecter une valeur à `dateEnregistrement` seulement en cas d'ajout (`POST`), on ajoute le test suivant

```
public function persist($data, array $context = [])
{
    if (($context['collection_operation_name'] ?? null)  === '
       post') {
         $dateTimeZone = new \DateTimeZone('Europe/Paris');
         $data->setDateEnregistrement(new \DateTime('now',
            $dateTimeZone));
    }
    $this->entityManager->persist($data);
    $this->entityManager->flush();
}
```

Symfony

Pour affecter une valeur à `dateEnregistrement` seulement en cas d'ajout (`POST`), on ajoute le test suivant

```
public function persist($data, array $context = [])
{
    if (($context['collection_operation_name'] ?? null)  === '
       post') {
         $dateTimeZone = new \DateTimeZone('Europe/Paris');
         $data->setDateEnregistrement(new \DateTime('now',
            $dateTimeZone));
    }
    $this->entityManager->persist($data);
    $this->entityManager->flush();
}
```

Remarque

`collection_operation_name` pour Collection operations et `item_operation_name` pour Item operations

Ajoutons l'annotation `@ApiSubresource` à l'attribut `adresses` pour récupérer la liste d'adresses d'une personne dont l'`id` est passé dans la route

```
namespace App\Entity;

use App\Repository\PersonneRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Annotation\ApiResource;
use ApiPlatform\Core\Annotation\ApiSubresource;

/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResource
 */
class Personne
{
     // contenu précédent
         /**
     * @ORM\ManyToMany(targetEntity=Adresse::class, inversedBy="personnes", cascade={"persist
        "})
     * @Groups({"personne:read", "personne:write"})
     * @ApiSubresource
     */
    private $adresses;
```

Ajoutons l'annotation `@ApiSubresource` à l'attribut `adresses` pour récupérer la liste d'adresses d'une personne dont l'`id` est passé dans la route

```
namespace App\Entity;

use App\Repository\PersonneRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Annotation\ApiResource;
use ApiPlatform\Core\Annotation\ApiSubresource;

/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResource
 */
class Personne
{
     // contenu précédent
         /**
     * @ORM\ManyToMany(targetEntity=Adresse::class, inversedBy="personnes", cascade={"persist
        "})
     * @Groups({"personne:read", "personne:write"})
     * @ApiSubresource
     */
    private $adresses;
```

Ainsi on a une nouvelle route pour le verbe `GET` : `/api/personnes/{id}/adresses`

@ApiFilter

- permet de filtrer le résultat de recherche
- peut concerner une entité ou un attribut
- peut utiliser plusieurs classes de filtre
  - `SearchFilter` : principalement pour les champs de type chaîne de caractères
  - `DateFilter` : pour les champs de type date
  - `NumericFilter` et `RangeFilter` : pour les champs de type numérique
  - ...

@ApiFilter

- permet de filtrer le résultat de recherche
- peut concerner une entité ou un attribut
- peut utiliser plusieurs classes de filtre
  - `SearchFilter` : principalement pour les champs de type chaîne de caractères
  - `DateFilter` : pour les champs de type date
  - `NumericFilter` et `RangeFilter` : pour les champs de type numérique
  - ...

Liste complète

https://api-platform.com/docs/core/filters/

Ajoutons l'annotation `@ApiFilter` à l'entité `Adresse` pour effectuer une recherche selon la ville (par exemple)

```
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use App\Repository\AdresseRepository;
use ApiPlatform\Core\Annotation\ApiFilter;
use Doctrine\Common\Collections\Collection;
use ApiPlatform\Core\Annotation\ApiResource;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;

/**
 * @ORM\Entity(repositoryClass=AdresseRepository::class)
 * @ApiResource
 * @ApiFilter(SearchFilter::class, properties={"ville": "partial"})
 */
class Adresse
```

Ajoutons l'annotation `@ApiFilter` à l'entité `Adresse` pour effectuer une recherche selon la ville (par exemple)

```
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use App\Repository\AdresseRepository;
use ApiPlatform\Core\Annotation\ApiFilter;
use Doctrine\Common\Collections\Collection;
use ApiPlatform\Core\Annotation\ApiResource;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;

/**
 * @ORM\Entity(repositoryClass=AdresseRepository::class)
 * @ApiResource
 * @ApiFilter(SearchFilter::class, properties={"ville": "partial"})
 */
class Adresse
```

Exemple de route pour le test : `/api/adresses?ville=mars`

Symfony

Valeurs possibles de `properties` de `SearchFilter`

- `partial` ≡ `ville like %texte%`
- `exact` ≡ `ville = texte`
- `end` ≡ `ville like %texte`
- `start` ≡ `ville like texte%`

Symfony

Valeurs possibles de `properties` de `SearchFilter`

- `partial` ≡ `ville like %texte%`
- `exact` ≡ `ville = texte`
- `end` ≡ `ville like %texte`
- `start` ≡ `ville like texte%`

Il est possible aussi d'effectuer la recherche à partir de la ressource `Personne` : `/api/personnes/9/adresses?ville=mars`

On peut aussi définir des contraintes de recherche sur les nombres en utilisant `RangeFilter`

```php
use Doctrine\ORM\Mapping as ORM;
use App\Repository\AdresseRepository;
use ApiPlatform\Core\Annotation\ApiFilter;
use Doctrine\Common\Collections\Collection;
use ApiPlatform\Core\Annotation\ApiResource;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\RangeFilter;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;

/**
 * @ORM\Entity(repositoryClass=AdresseRepository::class)
 * @ApiResource
 * @ApiFilter(SearchFilter::class, properties={"ville": "partial"})
 * @ApiFilter(RangeFilter::class, properties={"codePostal"})
 */
class Adresse
```

On peut aussi définir des contraintes de recherche sur les nombres en utilisant `RangeFilter`

```php
use Doctrine\ORM\Mapping as ORM;
use App\Repository\AdresseRepository;
use ApiPlatform\Core\Annotation\ApiFilter;
use Doctrine\Common\Collections\Collection;
use ApiPlatform\Core\Annotation\ApiResource;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\RangeFilter;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;

/**
 * @ORM\Entity(repositoryClass=AdresseRepository::class)
 * @ApiResource
 * @ApiFilter(SearchFilter::class, properties={"ville": "partial"})
 * @ApiFilter(RangeFilter::class, properties={"codePostal"})
 */
class Adresse
```

Exemple de route pour le test : `/api/adresses?ville=mars&codePostal[gt]=13006`

Symfony

Valeurs possibles de `properties` de `RangeFilter`

- `gt` ≡ supérieur à
- `lt` ≡ inférieur à
- `gte` ≡ supérieur ou égal à
- `lte` ≡ inférieur ou égal à
- `between` ≡ entre (Exemple : codePostal[between]=13000..13016)

On peut aussi définir des contraintes sur les entités imbriquées (inverses)

```
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use App\Repository\PersonneRepository;
use ApiPlatform\Core\Annotation\ApiFilter;
use Doctrine\Common\Collections\Collection;
use ApiPlatform\Core\Annotation\ApiResource;
use ApiPlatform\Core\Annotation\ApiSubresource;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;

/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResource(
 *     normalizationContext={"groups"={"personne:read"}},
 *     denormalizationContext={"groups"={"personne:write"}}
 * )
 * @ApiFilter(SearchFilter::class, properties={"adresses": "exact"})
 */
class Personne
```

On peut aussi définir des contraintes sur les entités imbriquées (inverses)

```php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;
use App\Repository\PersonneRepository;
use ApiPlatform\Core\Annotation\ApiFilter;
use Doctrine\Common\Collections\Collection;
use ApiPlatform\Core\Annotation\ApiResource;
use ApiPlatform\Core\Annotation\ApiSubresource;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;

/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResource(
 *     normalizationContext={"groups"={"personne:read"}},
 *     denormalizationContext={"groups"={"personne:write"}}
 * )
 * @ApiFilter(SearchFilter::class, properties={"adresses": "exact"})
 */
class Personne
```

Pour récupérer les personnes qui ont l'adresse avec un `id = 49` :
`/api/personnes?adresses=49`

On peut aussi chercher selon un attribut dans l'entité inverse autre que l'identifiant

```
use Doctrine\ORM\Mapping as ORM;
use App\Repository\PersonneRepository;
use ApiPlatform\Core\Annotation\ApiFilter;
use Doctrine\Common\Collections\Collection;
use ApiPlatform\Core\Annotation\ApiResource;
use ApiPlatform\Core\Annotation\ApiSubresource;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;

/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResource(
 *     normalizationContext={"groups"={"personne:read"}},
 *     denormalizationContext={"groups"={"personne:write"}}
 * )
 * @ApiFilter(SearchFilter::class, properties={"adresses.ville": "
   partial"})
 */
class Personne
```

On peut aussi chercher selon un attribut dans l'entité inverse autre que l'identifiant

```
use Doctrine\ORM\Mapping as ORM;
use App\Repository\PersonneRepository;
use ApiPlatform\Core\Annotation\ApiFilter;
use Doctrine\Common\Collections\Collection;
use ApiPlatform\Core\Annotation\ApiResource;
use ApiPlatform\Core\Annotation\ApiSubresource;
use Doctrine\Common\Collections\ArrayCollection;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;

/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResource(
 *     normalizationContext={"groups"={"personne:read"}},
 *     denormalizationContext={"groups"={"personne:write"}}
 * )
 * @ApiFilter(SearchFilter::class, properties={"adresses.ville": "
   partial"})
 */
class Personne
```

Pour récupérer les personnes qui ont une adresse dont le nom de la ville contient `mars` : `/api/personnes?adresses.ville=mars`

Symfony

Exercice 2

Créez une application **Angular** qui permet à un utilisateur, via des interfaces graphiques) la gestion de personnes (ajout, modification, suppression, consultation et recherche) en utilisant les web services définis par **Symfony**.

Symfony

JWT : JSON Web Token

- Librairie d'échange sécurisé d'informations
- Utilisant des algorithmes de cryptage comme HMAC SHA256 ou RSA
- Utilisant les jetons (tokens)
- Un jeton est composé de trois parties séparées par un point :
  - entête (header) : objet JSON décrivant le jeton encodé en base 64
  - charge utile (payload) : objet JSON contenant les informations du jeton encodé en base 64
  - Une signature numérique = concaténation de deux éléments précédents séparés par un point + une clé secrète (le tout crypté par l'algorithme spécifié dans l'entête)
- Documentation officielle : `https://jwt.io/introduction/`

Symfony

Entête : exemple

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Symfony

Entête : exemple

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Charge utile : exemple

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Symfony

Exemple de construction de signature en utilisant l'algorithme précisé dans l'entête

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Symfony

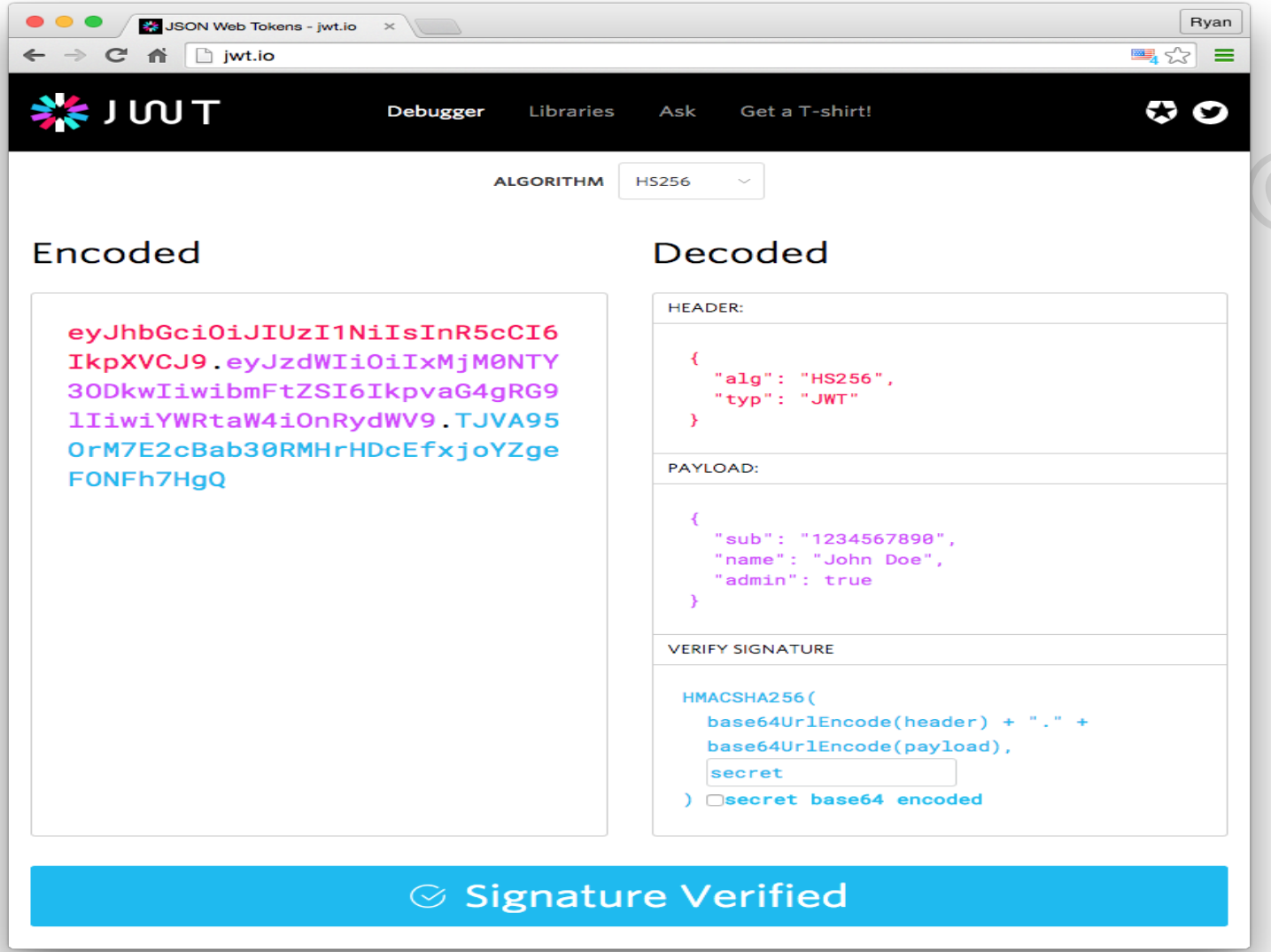
Exemple de construction de signature en utilisant l'algorithme précisé dans l'entête

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Résultat

Symfony

Exemple complet (pour tester `https://jwt.io/#debugger-io`)



Symfony

Plusieurs étapes

- Préparation de la partie utilisateur (qui va se connecter)
- Préparation de la partie JWT (clé publique, clé privée...)
- Gestion de rôles

Symfony

Intégrons le bundle de sécurité dans notre projet

```
composer require symfony/security-bundle
```

Symfony

Pour créer la classe `User`

- exécutez la commande `php bin/console make:user`
- répondez à `The name of the security user class` par `User`
- répondez à `Do you want to store user data in the database (via Doctrine)?` par `yes`
- répondez à `Enter a property name that will be the unique "display" name for the user` par `email`
- répondez à `Does this app need to hash/check user passwords?` par `yes`

Symfony

Pour créer la classe `User`

- exécutez la commande `php bin/console make:user`
- répondez à `The name of the security user class` par `User`
- répondez à `Do you want to store user data in the database (via Doctrine)?` par `yes`
- répondez à `Enter a property name that will be the unique "display" name for the user` par `email`
- répondez à `Does this app need to hash/check user passwords?` par `yes`

Le résultat est

```
created: src/Entity/User.php
created: src/Repository/UserRepository.php
updated: src/Entity/User.php
updated: config/packages/security.yaml
```

Nouveau contenu de `security.yaml`

```
security:
    encoders:
        App\Entity\User:
            algorithm: auto

    # https://symfony.com/doc/current/security.html#where-do-users-come
      -from-user-providers
    providers:
        # used to reload user from session & other features (e.g.
          switch_user)
        app_user_provider:
            entity:
                class: App\Entity\User
                property: email
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            anonymous: lazy
            provider: app_user_provider

    access_control:
```

Symfony

Pour créer la table `User`

- exécutez la commande `php bin/console make:migration`
- et ensuite la commande `php bin/console doctrine:migrations:migrate`

Symfony

Pour créer la table `User`

- exécutez la commande `php bin/console make:migration`
- et ensuite la commande `php bin/console doctrine:migrations:migrate`

Pour remplir la table `User` avec des données aléatoires

- installez le bundle de fixture `composer require --dev doctrine/doctrine-fixtures-bundle`
- demandez à ce bundle de créer une classe fixtures `php bin/console make:fixtures`
- répondez à `The class name of the fixtures to create` par `UserFixtures`

Symfony

Contenu généré pour `UserFixtures`

```php
namespace App\DataFixtures;

use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Persistence\ObjectManager;

class UserFixtures extends Fixture
{

    public function load(ObjectManager $manager)
    {
        // $product = new Product();
        // $manager->persist($product);

        $manager->flush();
    }
}
```

Symfony

Nouveau contenu de `UserFixtures`

```php
class UserFixtures extends Fixture
{
    private $passwordEncoder;

    public function __construct(UserPasswordEncoderInterface $passwordEncoder)
    {
        $this->passwordEncoder = $passwordEncoder;
    }
    public function load(ObjectManager $manager)
    {
        $user = new User();
        $user->setEmail('john@wick.us');
        $user->setRoles(['ROLE_ADMIN']);
        $user->setPassword($this->passwordEncoder->encodePassword($user, 'wick'));
        $manager->persist($user);
        $user2 = new User();
        $user2->setEmail('jack@dalton.us');
        $user2->setPassword($this->passwordEncoder->encodePassword($user2, 'dalton'));
        $manager->persist($user2);
        $manager->flush();
    }
}
```

Symfony

Nouveau contenu de `UserFixtures`

```
class UserFixtures extends Fixture
{
    private $passwordEncoder;

    public function __construct(UserPasswordEncoderInterface $passwordEncoder)
    {
        $this->passwordEncoder = $passwordEncoder;
    }
    public function load(ObjectManager $manager)
    {
        $user = new User();
        $user->setEmail('john@wick.us');
        $user->setRoles(['ROLE_ADMIN']);
        $user->setPassword($this->passwordEncoder->encodePassword($user, 'wick'));
        $manager->persist($user);
        $user2 = new User();
        $user2->setEmail('jack@dalton.us');
        $user2->setPassword($this->passwordEncoder->encodePassword($user2, 'dalton'));
        $manager->persist($user2);
        $manager->flush();
    }
}
```

Les `use` **nécessaires**

```
use App\Entity\User;
use Symfony\Component\Security\Core\Encoder\UserPasswordEncoderInterface;
```

Symfony

Pour insérer les utilisateurs dans la base de données, exécutez

```
php bin/console doctrine:fixtures:load ou php
bin/console d:f:l
```

Symfony

Intégrons le bundle **JWT** dans notre projet

- `composer require lexik/jwt-authentication-bundle`

Symfony

Pour créer les clés

- exécutez `mkdir config/jwt` (pour créer un répertoire `jwt` dans `config`)
- exécutez `openssl genrsa -out config/jwt/private.pem -aes256 4096` pour générer une clé privée (n'oubliez pas le passphrase)
- exécutez `openssl rsa -pubout -in config/jwt/private.pem -out config/jwt/public.pem` pour générer une clé public

Symfony

Pour créer les clés

- exécutez `mkdir config/jwt` (pour créer un répertoire `jwt` dans `config`)
- exécutez `openssl genrsa -out config/jwt/private.pem -aes256 4096` pour générer une clé privée (n'oubliez pas le passphrase)
- exécutez `openssl rsa -pubout -in config/jwt/private.pem -out config/jwt/public.pem` pour générer une clé public

Vérifiez le contenu suivant dans `.env` (remplacez passphrase par sa valeur)

```
###> lexik/jwt-authentication-bundle ###
JWT_SECRET_KEY=%kernel.project_dir%/config/jwt/private.pem
JWT_PUBLIC_KEY=%kernel.project_dir%/config/jwt/public.pem
JWT_PASSPHRASE=symfony
###< lexik/jwt-authentication-bundle ###
```

Modifiez la section `firewalls` **de** `security.yaml`

```
security:

    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        api:
            pattern: ^/ws/
            stateless: true
            anonymous: true
            provider: app_user_provider
            guard:
                authenticators:
                    - lexik_jwt_authentication.jwt_token_authenticator
        main:
            anonymous: true
            json_login:
                check_path: /authentication_token
                username_path: email
                password_path: password
                success_handler: lexik_jwt_authentication.handler.authentication_success
                failure_handler: lexik_jwt_authentication.handler.authentication_failure
            guard:
                authenticators:
                    - lexik_jwt_authentication.jwt_token_authenticator
    access_control:

        - { path: ^/ws/docs, roles: IS_AUTHENTICATED_ANONYMOUSLY } # Pour autoriser l'accès à
          Swagger UI
        - { path: ^/authentication_token, roles: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/, roles: IS_AUTHENTICATED_FULLY }
```

Symfony

Définissez la route `/authentication_token` **dans** `routes.yaml`

```
authentication_token:
    path: /authentication_token
    methods: ['POST']
```

Symfony

Pour exiger le rôle `ROLE_ADMIN` aux demandeurs de la ressource `Personne`

```php
namespace App\Entity;

use App\Repository\PersonneRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Annotation\ApiResource;

/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResource(
 *     attributes={"security"="is_granted('ROLE_ADMIN')"},
 *     itemOperations={"get", "put", "delete"},
 *     normalizationContext={"groups"={"personne:read"}},
 *     denormalizationContext={"groups"={"personne:write"}}
 * )
 */
class Personne
```

Symfony

Pour obtenir le jeton avec **POSTMAN**

- Dans la liste déroulante, choisir `POST` puis saisir l'url vers notre web service `http://localhost:8000/authentication_token`
- Dans le `Headers`, saisir `Content-Type` comme `Key` et `application/json` comme `Value`
- Ensuite cliquer sur `Body`, cocher `raw`, choisir `JSON (application/json)` et saisir des données sous format `JSON` correspondant à l'objet personne à ajouter

```
{
    "email": "wick@wick.us",
    "password": "wick"
}
```

- Cliquer sur `Send` puis copier le jeton

Symfony

Pour obtenir la liste des personnes avec **POSTMAN**

- Dans la liste déroulante, choisir `GET` puis saisir l'url vers notre web service `http://localhost:8000/ws/personnes`
- Dans le `Headers`, saisir `Content-Type` comme `Key` et `application/json` comme `Value`
- Dans `Authorization`, cliquer sur `Type` et choisir `raw`, choisir `Bearer Token` et coller le token
- Cliquer sur `Send`

Symfony

Il est possible de sécuriser l'accès seulement à certaines méthodes (mais pas toutes)

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResource(
 *      collectionOperations={
 *          "get",
 *          "post"={"security"="is_granted('ROLE_ADMIN')"}
 *      },
 *      itemOperations={
 *           "get",
 *           "put"={"security"="is_granted('ROLE_ADMIN')"},
 *           "delete"={"security"="is_granted('ROLE_ADMIN')"}
 * },
 *      normalizationContext={"groups"={"personne:read"}},
 *      denormalizationContext={"groups"={"personne:write"}}
 * )
 */
class Personne
```

Symfony

Il est possible de sécuriser l'accès seulement à certaines méthodes (mais pas toutes)

```
/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 * @ApiResource(
 *      collectionOperations={
 *          "get",
 *          "post"={"security"="is_granted('ROLE_ADMIN')"}
 *      },
 *      itemOperations={
 *           "get",
 *           "put"={"security"="is_granted('ROLE_ADMIN')"},
 *           "delete"={"security"="is_granted('ROLE_ADMIN')"}
 * },
 *      normalizationContext={"groups"={"personne:read"}},
 *      denormalizationContext={"groups"={"personne:write"}}
 * )
 */
class Personne
```

N'oublions pas de commenter le contenu de la section `access_control` dans `security.yaml`